Försättsblad till skriftlig tentamen vid Linköpings universitet

Datum för tentamen	2016-06-04
Sal (1)	U1
Tid	14-18
Kurskod	TDDD48
Provkod	TEN1
Kursnamn/benämning **Provnamn/benämning**	Automatisk planering Skriftlig tentamen
Institution	IDA
Antal uppgifter som ingår i tentamen	5
Jour/Kursansvarig Ange vem som besöker salen	Jonas Kvarnström
Telefon under skrivtiden	0704-737579
Besöker salen ca klockan	Om möjligt - ring gärna istället
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund, ankn. 2362, anna.grabska.eklund@gmail.com
Tillåtna hjälpmedel	inga
Övrigt	
Antal exemplar i påsen	

TDDD48 Automated Planning 2016-06-04

Important Instructions: Read before you begin!

- Though the questions are in English, feel free to **answer in Swedish** if you prefer! Det går bra att skriva på svenska!
- *Clear and comprehensible* explanations and motivations are always required. This does not necessarily mean that each answer should be a long essay. What is important is that all the relevant facts are present and clearly explained.
- Make an effort to write explanations that can be understood by someone who *does not already know* the correct answer. If your answer explains a topic well enough that a fellow student could *learn* something new from the answer and/or apply it in practice, then you have probably succeeded. Conversely, if we ask how HTNs work, saying that "there are methods and operators and you construct a tree using patterns and there can be constraints too, and there's no goal" does not provide much in terms of *useful* information, and certainly will not give you a full score.
- To see if you have succeeded, turn the tables and **try to misunderstand.** Specifically, after you write an answer, go back and see if you can creatively misinterpret the answer. If so, *clarify, clarify, clarify*!
- Concrete examples or counterexamples may be useful as part of a motivation. If so, please make sure you include all relevant information about the example you have chosen to use. What is relevant naturally depends on how you use the example.
- When asked to provide examples or illustrate something through a planning problem instance, save time by *keeping the example as small as possible.*

1 General Concepts in Automated Planning

As you know, *lifted* planning is a general technique applicable to a variety of search spaces and planning algorithms. During the lectures, we discussed this mostly in the context of *lifted partial-order planning*, but since the technique itself is general, it could also be applied to for example backward planning or HTN planning.

Explain what lifted planning is in general. Then give a concrete **example** of why it can be useful when generating partial-order plans in particular. In this example you must:

- **Show/illustrate** a small lifted partial-order plan structure, or a sufficiently informative fragment of one that makes it clear that you know how partial order plans are structured.
- **Contrast** it against an alternative *non-lifted* partial-order plan structure. This plan structure must also be clearly illustrated. Alternatively, show clearly and concretely how the lifted plan you already illustrated would differ if it was non-lifted.
- **Explain** what is better about the lifted version: In which way does it "help" you to use a lifted plan when you are searching for a solution? **(3 points)**

2 SATisfiability-based Planning

We will now consider planning based on propositional satisfiability ("SAT planning").

a) A SAT planner translates a planning problem into a set of boolean propositions and a set of formulas over these propositions. Typically, there are two specific types of information encoded as boolean propositions. In other words, once a SAT solver has found a solution, two specific types of information can be inferred from the values of the boolean propositions. Which ones? **(2 points)**

b) SAT *planners* must be able to find solution plans of arbitrary length, where the length is not known in advance. In contrast, the general SAT *solvers* that they are based on require an input containing a fixed and finite number of boolean propositions. How is this discrepancy handled by a SAT planner? **(2 points)**

c) SAT planners require *frame axioms* (for example, *explanatory* frame axioms) to ensure that the SAT solver gives correct results that correspond to valid plans.

Explain how planning could go wrong if frame axioms were not included in the SAT translation of a planning problem. Also demonstrate using a concrete example. The example does not have to involve an entire SAT assignment and SAT formula processed from beginning to end, as this would take considerable time to generate by hand. It is sufficient to illustrate the most central aspects of how the planning process could return invalid plans given a lack of frame axioms. **(2 points)**

3 Heuristics and Search Guidance

a) **Landmarks.** We use the logistics domain (Appendix A) with the following problem instance:

- There are two packages $\{p_1, p_2\}$, two trucks $\{t_1, t_2\}$ and three locations $\{l_1, l_2, l_3\}$, none of which are airports.
- $s_0 = \{\mathsf{at}(p_1, l_1), \mathsf{at}(p_2, l_2), \mathsf{at}(t_1, l_2), \mathsf{at}(t_2, l_2)\} \cup \{\mathsf{same\text{-}city}(l, l') \mid l, l' \in \{l_1, l_2, l_3\}\}$, where the latter part indicates that all locations are in the same city. No packages are loaded into vehicles and we do not use type predicates.
- The goal is $g = \{\mathsf{at}(p_1, l_3), \mathsf{at}(p_2, l_3), \mathsf{at}(t_1, l_2), \mathsf{at}(t_2, l_2)\}$.

Your task:

- Explain: What is a *disjunctive action landmark*?
- Show three distinct (disjunctive or non-disjunctive) action landmarks for the problem instance above.
- Explain *why* these are landmarks. The explanation should not repeat the definition of a landmark. Instead it must show why the landmarks *satisfy* the definition.

(3 points)

b) **Pattern databases.** Describe the general ideas behind standard pattern database (PDB) heuristics, assuming that a *single* pattern is used. Specifically:

- A fundamental property of PDB heuristics is the use of a pre-processing phase that is specific to a problem instance but is performed before search begins. Assume as given a problem instance $\mathscr{P} = \langle O, s_0, g \rangle$ in the state variable representation, with no additional PDB-specific information provided. Describe what the pre-processing phase does with this problem instance, step by step. Also describe what the resulting output will consist of.
- How is a heuristic value $h(s)$ calculated when a new state s is encountered during search, if we use the standard PDB heuristic (still using a single pattern)? How is the result of the pre-processing phase used during this calculation?

(3 points)

c) **Combining heuristics.** There are several techniques for "combining" a set of admissible heuristics into a new and stronger, but still admissible, heuristic function.

Describe a technique that is applicable for *every* set of admissible heuristics, regardless of how these heuristics are computed. Explain why the resulting value is admissible.
(2 points)

4 Planning with Incomplete Information

a) Explain the meaning of *non-observable* planning. (For example, what isn't observable? What do you know, what don't you know, and when?) **(1 point)**

b) Explain the meaning of *non-deterministic* planning and *probabilistic* planning, and how these two forms of planning differ from classical forms of planning.

Your explanation could be based on how the associated state transition systems differ and/or on a more "intuitive" explanation. In the latter case, remember that explanations must still be sufficiently clear that someone who is not familiar with the concepts *learns* what the concepts mean. **(2 points)**

5 Motion Planning

In motion planning, what is the *workspace* and what is the *configuration space*? Explain this on a conceptual level. Also give concrete examples: What might the workspace and configuration space be for a typical *car*, for a *humanoid robot*, or for some other non-trivial entity? **(2 points)**

A The Logistics Domain

The **logistics domain** contains a set of *packages* that should be transported to their destinations. A package can be transported by *truck* between any two *locations* in the same city, and by *airplane* between special *airport* locations in different cities. Since trucks cannot deliver packages directly to other cities, and airplanes cannot visit arbitrary locations, delivering a package might require using a truck to move it to an airport, using an airplane to move it to another city and then once again using a truck to get the package to its final destination.

We assume the following types of objects:

- thing, with subtypes package and vehicle
- vehicle, with subtypes truck and airplane
- location, with subtype airport

A model of this domain may include the following **operators**.

- load-truck(package, truck, location) loads a package into a truck, given that they are both at the same location. The truck can hold an arbitrary number of packages.
- unload-truck(package, truck, location) unloads a package from a truck in the current location.
- load-plane(package, airplane, location) loads a package into a plane, given that they are both at the same location. The plane can hold an arbitrary number of packages.
- unload-plane(package, airplane, location) unloads a package from an airplane in the current location.
- drive-truck(truck, location, location) drives a truck between locations in the same city.
- fly-plane(plane, airport, airport) flies a plane between two airports in different cities.

We assume the following **predicates** are available:

- at(thing,loc) – the package or vehicle thing is at the location loc. Note that a package "at" a certain location cannot be "in" a vehicle in the same state.
- in(x,vehicle) – the package x is in the given vehicle.
- same-city(loc1,loc2) – the given locations are in the same city. Note that a location must be in the same location as itself.

We assume a **typed** domain model where each operator parameter and predicate parameter is given a specific type and cannot take on values outside that type. For example, if t_1 is a truck, an expression such as $\mathsf{at}(t_1, t_1)$ is not merely false but incorrect. Nevertheless, we provide the following **type predicates** that may be useful in some situations: thing(x), package(x), vehicle(x), truck(x), airplane(x), location(x) and airport(x).

We can then define the operators more formally as shown on the following page:

- load-truck(package *pkg*, truck *trk*, location *loc*)
  Precondition: at(*pkg*, *loc*) $\wedge$ at(*trk*, *loc*)
  Effects: $\neg$at(*pkg*, *loc*), in(*pkg*, *trk*)

- load-airplane(package *pkg*, airplane *plane*, location *loc*)
  Precondition: at(*pkg*, *loc*) $\wedge$ at(*plane*, *loc*)
  Effects: $\neg$at(*pkg*, *loc*), in(*pkg*, *plane*)

- unload-truck(package *pkg*, truck *trk*, location *loc*)
  Precondition: in(*pkg*, *trk*) $\wedge$ at(*trk*, *loc*)
  Effects: $\neg$in(*pkg*, *trk*), at(*pkg*, *loc*)

- unload-airplane(package *pkg*, airplane *plane*, location *loc*)
  Precondition: in(*pkg*, *plane*) $\wedge$ at(*plane*, *loc*)
  Effects: $\neg$in(*pkg*, *plane*), at(*pkg*, *loc*)

- drive-truck(truck *trk*, location *from*, location *to*)
  Precondition: at(*trk*, *from*) $\wedge$ same-city(*from*, *to*)
  Effects: $\neg$at(*trk*, *from*), at(*trk*, *to*)

- fly-airplane(airplane *plane*, airport *from*, airport *to*)
  Precondition: at(*plane*, *from*)
  Effects: $\neg$at(*plane*, *from*), at(*plane*, *to*)